Problem A. Letters from Parts

Input file:	`standard input`
Output file:	`standard output`
Time limit:	1 second
Memory limit:	1024 megabytes

Concerned about the dominance of the C++ and other compiled languages in programming competitions, representatives of the National Association of Intepreters (NAI) decided to hold a championship where only Bash, Basic and Python would be allowed.

Before the championship final, the organizers ordered an advertisement on the building where the final is planned to take place. The advertisement must consist of a string of English letters N, A, and I. The letter N is made from two vertical and one diagonal stick, the letter I is made of one vertical stick, and the letter A is made from two diagonal sticks and one horizontal stick. The elements of the inscription can be reflected 180 degrees, but they cannot be rotated (so the vertical, diagonal and horizontal sticks cannot be transformed one into another, but the / can be transformed to \ and vice versa).

The supplier delivered the inscription in the form of sets of parts, and some of them may have been lost. Given this list of parts, determine whether it can be definitively established that a part is lost (i.e., it is impossible to assemble the letters so that all parts are used). Otherwise, find the maximum number of "NAI" strings that can be obtained in the assembled inscription (the letters N, A, and I must go consecutively; for example, in the inscription "NAINNAI" there are two such strings, while in the inscription "NNAAII" there are none).

Input

The first line of input contains one integer v — the number of vertical sticks ($0 \le v \le 10^5$).

The second line of input contains one integer d — the number of diagonal sticks ($0 \le d \le 10^5$).

The third line of input contains one integer h — the number of horizontal sticks ($0 \le h \le 10^5$).

Output

If any part is lost and there are extra elements in the inscription that cannot be assembled into letters — output -1.

Otherwise, output one integer — the maximum number of "NAI" strings that can be obtained in the assembled inscription.

Examples

standard input	standard output
1 1 1	-1
0 0 0	0
10 6 2	2

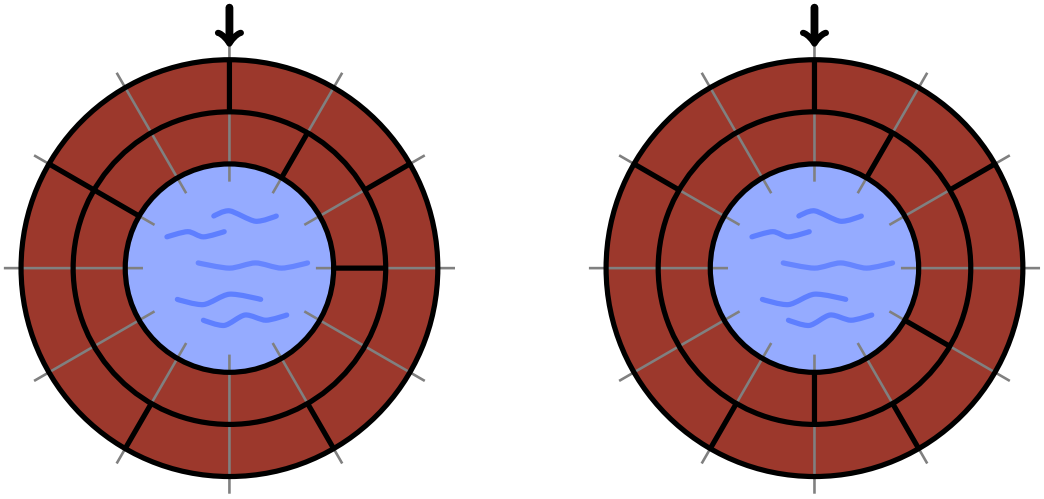
Problem B. Build Well

Input file:	*standard input*
Output file:	*standard output*
Time limit:	4 seconds
Memory limit:	1024 megabytes

Bob the Builder is tired of building tiny houses and paving narrow roads, and he strives for something bigger. The new job given to him by a very eccentric client is exactly what he needs: He is tasked with building a round well of a certain circumference that is infinitely deep! His client assured him that he does not need to worry about the building material, and that an infinite supply of various kinds of bricks, only differing by their arc lengths, has already been ordered for him.

Of course, building a stable well takes very careful planning, especially if it is supposed to be infinitely deep. In particular, a well is only stable if no two gaps between bricks in consecutive rows end up directly above each other, as shown in the figure below. The bricks are all of integer length and can only be offset by an integer length. Note that even a brick that covers the complete circumference has a start and an end and therefore, a gap.

Bob knows from his long-time experience that if it is possible to build such a well, then it can be done by alternating just two row configurations.



On the left, we see an unstable well using the brick types of Sample Input 1. On the right, we see a stable well using the same brick types. For visual reasons, bricks on the outer rows appear larger even though they have the same total arc length. Note that even though only two rows of the well are shown, it is possible to build an infinitely deep well by repeating these two row configurations. The arrow points to the zero offset of the sample.

Bob is terribly excited about the new job and quickly goes to work. Given the types of arc bricks available, is it possible to build a stable well of circumference exactly w and infinite height? If yes, how should Bob build it using only two alternating row configurations?

Input

The input consists of one line with two integers n and w ($1 \le n, w \le 3 \cdot 10^5$) — the number of brick types and the circumference of the well. The following line contains n integers b_i ($1 \le b_i \le w$) — the arc lengths of the brick types.

Note that Bob has an infinite supply of all brick types.

Output

If it is possible to build a well, output `possible`. Otherwise, output `impossible`.

If a well can be built, provide two row configurations that can be used in an alternating fashion. For both row configurations, first output the number of bricks needed for that row and the clockwise offset where

to place the first brick in a single line, followed by a line containing the arc lengths of the bricks in the clockwise order you want to use them. Note that the offset has to be a non negative integer smaller than the circumference of the well. Your solution is considered valid if alternating the two rows infinitely would result in a stable well.

If there are multiple valid solutions, you may output any one of them.

Examples

standard input	*standard output*
4 12 3 2 7 2	possible 5 0 2 3 2 3 2 3 1 3 2 7
3 11 6 7 8	impossible

Problem C. Centrifuge

Input file:	*standard input*
Output file:	*standard output*
Time limit:	1 second
Memory limit:	1024 megabytes

On your latest intergalactic scavenger trip, you discovered an abandoned futuristic robot. Mysteriously, the robot consists of n ball-shaped joints with some kind of fluid inside. These joints are all connected together via $n-1$ flexible pipes that allow the fluid to flow bidirectionally from one joint to another. To analyze this strange fluid inside the robot, you decide to use a centrifuge.

Since you have no clue how to actually use a centrifuge, you secure one of the joints to the center of it and turn the machine on. The machine then rapidly spins around the center, and all the fluids are pushed away from the center towards the outside. Whenever the fluid has more than one pipe to flow through, the fluid splits evenly between all of these pipes.

You wonder how much fluid will be in the outermost joints at the end of this process. After thinking so much, you forgot which joint you secured to the center. Therefore, you have to calculate the expected amount of fluid in each joint as if you chose a joint at random.

Input

The first line contains one integer n ($1 \le n \le 2 \cdot 10^5$) — the number of joints of the robot.

The next line contains n integers a_i ($0 \le a_i \le 10^9$) — the amount of fluid in the i-th joint.

The next $n-1$ lines contain two integers u and v ($1 \le u, v \le n$) — indicating a pipe between joint u and v. These pipes form a tree.

Output

Output n lines with one integer per line, the i-th representing the expected amount of fluid in the i-th joint after the process modulo 998 244 353.

Formally, let $M = 998\,244\,353$. It can be shown that the answer can be expressed as an irreducible fraction $\frac{p}{q}$, where p and q are integers and $q \not\equiv 0 \pmod{M}$. Output the integer equal to $p \cdot q^{-1} \bmod M$. In other words, output such an integer x that $0 \le x < M$ and $x \cdot q \equiv p \pmod{M}$.

Examples

standard input	*standard output*
3 1 2 4 1 2 2 3	3 0 4
6 4 1 3 11 7 2 1 4 2 3 6 2 2 4 4 5	928921837 0 818005794 0 679360751 568444705

Note

In the first sample, if the first joint is fixed at the center, all fluid will flow from joint 1 to 2, and the combined fluid will flow to joint 3. In total, joint 1 and 2 will be empty, and all 7 units of fluid will be at joint 3.

If the second joint is fixed to the center, half of its fluid will flow to 1 and half will flow to 3. There will be 2, 0, and 5 units of fluid at each joint respectively. If joint 3 is fixed, all 7 units of fluid will be at joint 1.

In total, the expected amount of fluid in each joint is

$$\frac{1}{3} \cdot (0 + 2 + 7) = 3$$
$$\frac{1}{3} \cdot (0 + 0 + 0) = 0$$
$$\frac{1}{3} \cdot (7 + 5 + 0) = 4$$

Problem D. Infinity Triples

Input file:	*standard input*
Output file:	*standard output*
Time limit:	2 seconds
Memory limit:	1024 megabytes

Consider numbers in base b where all digits are equal to a with $1 \le a < b$. We call a triple (n, a, b) an *infinity triple* if infinitely many of those numbers are divisible by n.

For example, $(3, 9, 10)$ is an infinity triple because infinitely many of the numbers 9, 99, 999, ... are divisible by 3. The triple $(7, 9, 10)$ is also an infinity triple, but $(5, 9, 10)$ is not.

Given m, count the number of infinity triples with $1 \le n \le m$ and $1 \le a < b \le m$.

Input

The input contains one integer m ($2 \le m \le 10^5$).

Output

Output one integer, the number of infinity triples with $1 \le n \le m$ and $1 \le a < b \le m$.

Examples

standard input	*standard output*
2	1
3	6
42	25055

Note

In the first sample, $(1, 1, 2)$ is the only infinity triple.

In the second sample, the infinity triples are $(1, 1, 2)$, $(1, 1, 3)$, $(1, 2, 3)$, $(2, 1, 3)$, $(2, 2, 3)$, and $(3, 1, 2)$.

Problem E. Taxi

Input file:	*standard input*
Output file:	*standard output*
Time limit:	5 seconds
Memory limit:	1024 megabytes

There are n cities that are connected by $n-1$ roads, forming a tree. Note that each road has a given length.

When you are at city v, you can take a taxi of the local taxi company to any other city w. For this, you have to pay $a_v + d \cdot b_v$ cookies, where d is the distance from v to w. In other words, you have to pay the base cost a_v and additionally b_v for each unit of distance traveled.

You are currently at city 1, and for each other city v, you want to know the minimum cost to get there.

Input

The first line contains one integer n ($2 \le n \le 10^5$) — the number of cities.

The second line contains n integers a_i ($0 \le a_i \le 10^{12}$) — the base costs of the taxis.

The third line contains n integers b_i ($1 \le b_i \le 10^6$) — the cost per distance.

Then $n-1$ lines follow, describing the roads between the cities. Every line contains three integers u, v, and ℓ ($1 \le u, v \le n$, $u \ne v$, $1 \le \ell \le 10^6$) describing a bidirectional road between cities u and v of length ℓ.

Output

Output a single line containing $n-1$ integers. The i-th of them should be the minimum cost to get to city $i+1$.

Examples

standard input	*standard output*
3 0 1 2 8 4 4 1 2 1 1 3 7	8 41
2 353 313 928248 475634 2 1 898027	833591767049

Note

Consider the cost to get to city 3 in the first sample: Driving directly from 1 to 3 would cost $0+7\cdot 8 = 56$. It is better to drive from 1 to 2 with a cost of 8 and take a second taxi from 2 to 3 with a cost of $1+8\cdot 4 = 33$. While the distance traveled is larger, the cost is still smaller.

Problem F. Periodic Sequence

Input file:	*standard input*
Output file:	*standard output*
Time limit:	1 second
Memory limit:	1024 megabytes

As you may know, some fractions of integers $\frac{A}{B}$ result in an infinite periodic decimal representation. For example, $\frac{4}{7}$ results in $0.57\overline{142857}$, which means that the 142857 part is repeating itself. Note that we could also write $0.5714\overline{285714}$ or $0.571\overline{428571428571}$. As you see, it is not trivial to check if two such sequences are equal.

Therefore, you have to help us. You are given only the periodic part of two sequences and need to check if they are equal. Note that the periodic parts are considered equal if they can be made equal by repetition and cyclic shifting.

Input

The first line contains two integers n and m ($1 \le n, m \le 5 \cdot 10^5$) — the length of the first and second sequence.

The second line contains n integers a_i ($0 \le a_i < 10$) — the first sequence.

The third line contains m integers b_i ($0 \le b_i < 10$) — the second sequence.

Output

Print `YES` if the two sequences are equal and `NO` if they are not.

Examples

standard input	*standard output*
6 3 1 5 6 1 5 6 6 1 5	YES
7 3 1 5 6 1 5 6 7 5 6 7	NO

Problem G. Operations on Subarrays

Input file:	`standard input`
Output file:	`standard output`
Time limit:	1 second
Memory limit:	1024 megabytes

For any two positive integers, the operation BSw is defined as follows:

- Consider the binary representations of the two numbers.
- Choose a position in the binary representation of the first number.
- Swap the digit at that position in the first number with the digit at the same position in the second number.

You are given a sequence a. We define a *subarray* of this sequence, specified by two numbers l and r, as the sequence containing all elements from the l-th to the r-th inclusive, in the same order as in the original sequence.

Your task is to answer queries of the following type: for a given subarray, compute the minimum sum of squares of the numbers in the subarray after applying the BSw operation any number of times to any numbers in the subarray. Note that the queries are independent in the sense that the elements of the sequence do not change between queries (i.e., all operations for minimizing the sum are local and do not affect subsequent queries).

Input

The first line of input contains two integers n and q ($1 \le n \le 10^5$, $1 \le q \le 10^5$): the number of elements in the sequence and the number of queries, respectively. The second line contains n positive integers a_i ($1 \le a_i \le 10^6$), the elements of the sequence.

This is followed by q lines, each describing one query and containing two integers l and r ($1 \le l \le r \le n$) — the leftmost and rightmost elements of the subarray.

Note that all operations are applied to the original sequence, without applying changes caused by previous queries.

Output

For each query, output a single integer: the answer to that query.

Example

standard input	standard output
8 8 1 74 98 73 42 15 26 106 1 5 2 4 2 7 6 6 6 8 2 6 2 6 5 6	17334 20317 19590 225 8201 19074 19074 1989

Problem H. Mod Graph

Input file:	*standard input*
Output file:	*standard output*
Time limit:	1 second
Memory limit:	1024 megabytes

You are given an undirected, connected graph with n vertices. Every vertex v has a counter that operates modulo b_v. The initial state of that counter is a_v. Every time you visit that vertex, it is increased by one (i.e. $a_v \leftarrow (a_v + 1) \bmod b_v$).

You have to process q queries of the following two types:

- "`1` s": Suppose that you start at vertex s, you have to answer whether it is possible to make $a_v = 0$ for all v. You are allowed to take an arbitrary walk through G, i.e. you can use edges and vertices multiple times. Note that starting the walk at s already counts as visiting s, i.e. its counter is increased by one initially.
- "`2` v x": Update $a_v \leftarrow x$.

Input

The first line contains three integers n, m, and q ($1 \le n, q \le 5 \cdot 10^4$, $0 \le m \le 10^5$) — the number of vertices, edges, and queries, respectively.

The second line contains n integers $a_1, \ldots, a_n$.

The third line contains n integers $b_1, \ldots, b_n$ ($0 \le a_v < b_v \le 10^9$).

The following m lines contain the edges of the graph. Each of those m lines contains two integers u and v ($1 \le u, v \le n$, $u \ne v$) describing an edge connecting vertices u and v. The given graph is connected.

Finally, there are q lines describing the queries. They can be in the two formats described above, i.e.

- "`1` s": ($1 \le s \le n$)
- "`2` v x": ($1 \le v \le n$, $0 \le x < b_v$)

Output

For each query of type `1`, output `YES` in one line if it is possible to make $a_v = 0$ for all v and `NO` otherwise.

Example

standard input	*standard output*
2 1 4 0 0 3 3 1 2 1 1 2 1 1 1 1 1 2	YES NO YES

Note

In the first query, we start at vertex 1 with counter states $[1, 0]$.

We move along the walk $1 \to 2 \to 1 \to 2 \to 1 \to 2$.

The counter states change as follows: $[1, 0] \to [1, 1] \to [2, 1] \to [2, 2] \to [0, 2] \to [0, 0]$.

Problem I. Arithmetic Progression and Triangle

Input file:	`standard input`
Output file:	`standard output`
Time limit:	1 second
Memory limit:	1024 megabytes

It is known that in triangle ABC, the measures of the angles, expressed in degrees, form an arithmetic progression. You are given the measure of the smallest of the three angles. Determine whether such a triangle exists, and if it does, what the maximum and minimum values of the middle angle can be, as well as the maximum angle.

Input

The first line of the input contains one integer a ($1 \le a \le 178$) — the measure of the smallest of the three angles of the triangle.

Output

If a triangle with the specified properties does not exist, output -1.

Otherwise, in the first line, output two integers b_{min} and b_{max} — the maximum and minimum values of the middle angle. In the second line, output two integers c_{min} and c_{max} — the maximum and minimum values of the largest angle.

Examples

standard input	standard output
60	60 60 60 60
61	-1

Problem J. Permutation Recovery

Input file:	*standard input*
Output file:	*standard output*
Time limit:	2 seconds
Memory limit:	1024 megabytes

Initially, we had k permutations of the integers from 1 to n. We created a $2k \times n$ matrix by writing each permutation as well as its inverse in its own row. However, we forgot the permutations, and someone shuffled every column. Given this matrix, can you determine any set of permutations which we could have started with?

Input

The first line contains two integers n and k ($1 \le n \le 4 \cdot 10^4$, $1 \le k \le 7$).

The i-th of the following $2k$ lines contains n integers a_{ij}, the i-th row of the matrix ($1 \le a_{ij} \le n$).

It is guaranteed that the matrix could have been obtained as described above.

Output

Output k lines. Each of them should contain a permutation of the integers from 1 to n. After writing these permutations as well as their inverses in the rows of a $2k \times n$ matrix, it must be possible to obtain the input matrix by reordering the values in every column.

If there are multiple solutions, output any of them.

Examples

standard input	*standard output*
3 1 1 2 3 1 2 3	1 2 3
4 2 1 1 3 4 4 3 2 1 2 4 2 4 1 3 3 2	1 3 2 4 4 1 3 2